Lab_10_SpamEmail.m

```
%% Machine Learning
% Lab 10: Spam Classification with SVMs
% —— Water overflow ——
%{
In the first half of the exercise, you will implement regularized linear
regression to predict the amount of water flowing out of a dam using the
change of water level in a reservoir. In the next half, you will go through
some diagnostics of debugging learning algorithms and examine the effects
of bias v.s. variance.
%}

%% Initialization
clear ; close all; clc

%% ==================== Part 1: Email Preprocessing ====================
%  To use an SVM to classify emails into Spam v.s. Non—Spam, you first need
%  to convert each email into a vector of features. In this part, you will
%  implement the preprocessing steps for each email. You should
%  complete the code in processEmail.m to produce a word indices vector
%  for a given email.

fprintf('\nPreprocessing sample email (emailSample1.txt)\n');

% Extract Features
file_contents = readFile('emailSample1.txt');
word_indices  = processEmail(file_contents);

% Print Stats
fprintf('Word Indices: \n');
fprintf(' %d', word_indices);
fprintf('\n\n');

fprintf('Program paused. Press enter to continue.\n');
pause;

%% ==================== Part 2: Feature Extraction ====================
%  Now, you will convert each email into a vector of features in R^n.
%  You should complete the code in emailFeatures.m to produce a feature
%  vector for a given email.

fprintf('\nExtracting features from sample email (emailSample1.txt)\n');

% Extract Features
file_contents = readFile('emailSample1.txt');
word_indices  = processEmail(file_contents);
features      = emailFeatures(word_indices);

% Print Stats
fprintf('Length of feature vector: %d\n', length(features));
fprintf('Number of non—zero entries: %d\n', sum(features > 0));

fprintf('Program paused. Press enter to continue.\n');
pause;

%% =========== Part 3: Train Linear SVM for Spam Classification ========
%  In this section, you will train a linear classifier to determine if an
%  email is Spam or Not–Spam.

% Load the Spam Email dataset
% You will have X, y in your environment
load('spamTrain.mat');

fprintf('\nTraining Linear SVM (Spam Classification)\n')
fprintf('(this may take 1 to 2 minutes) ...\n')

C = 0.1;
model = svmTrain(X, y, C, @linearKernel);

p = svmPredict(model, X);

fprintf('Training Accuracy: %f\n', mean(double(p == y)) ∗ 100);

%% =================== Part 4: Test Spam Classification ================
%  After training the classifier, we can evaluate it on a test set. We have
%  included a test set in spamTest.mat

% Load the test dataset
% You will have Xtest, ytest in your environment
load('spamTest.mat');

fprintf('\nEvaluating the trained Linear SVM on a test set ...\n')

p = svmPredict(model, Xtest);

fprintf('Test Accuracy: %f\n', mean(double(p == ytest)) ∗ 100);
pause;


%% ================= Part 5: Top Predictors of Spam ====================
%  Since the model we are training is a linear SVM, we can inspect the
%  weights learned by the model to understand better how it is determining
%  whether an email is spam or not. The following code finds the words with
%  the highest weights in the classifier. Informally, the classifier
%  'thinks' that these words are the most likely indicators of spam.
%

% Sort the weights and obtin the vocabulary list
[weight, idx] = sort(model.w, 'descend');
vocabList = getVocabList();

fprintf('\nTop predictors of spam: \n');
for i = 1:15
    fprintf(' %–15s (%f) \n', vocabList{idx(i)}, weight(i));
end

fprintf('\n\n');
fprintf('\nProgram paused. Press enter to continue.\n');
pause;

%% =================== Part 6: Try Your Own Emails =====================
%  Now that you've trained the spam classifier, you can use it on your own
%  emails! In the starter code, we have included spamSample1.txt,
%  spamSample2.txt, emailSample1.txt and emailSample2.txt as examples.
%  The following code reads in one of these emails and then uses your
%  learned SVM classifier to determine whether the email is Spam or
%  Not Spam

% Set the file to be read in (change this to spamSample2.txt,
% emailSample1.txt or emailSample2.txt to see different predictions on
% different emails types). Try your own emails as well!
filename = 'spamSample1.txt';

% Read and predict
file_contents = readFile(filename);
word_indices  = processEmail(file_contents);
x             = emailFeatures(word_indices);
p = svmPredict(model, x);

fprintf('\nProcessed %s\n\nSpam Classification: %d\n', filename, p);
fprintf('(1 indicates spam, 0 indicates not spam)\n\n');
```

gaussianKernel.m

```matlab
function sim = gaussianKernel(x1, x2, sigma)
%RBFKERNEL returns a radial basis function kernel between x1 and x2
%   sim = gaussianKernel(x1, x2) returns a gaussian kernel between x1 and x2
%   and returns the value in sim

% Ensure that x1 and x2 are column vectors
x1 = x1(:); x2 = x2(:);

% You need to return the following variables correctly.
sim = 0;

sim = exp(-(sum((x1-x2).^2)/(2*sigma^2)));

end
```

dataset3Params.m

```matlab
function [C, sigma] = dataset3Params(X, y, Xval, yval)
%DATASET3PARAMS returns your choice of C and sigma for Part 3 of the exercise
%where you select the optimal (C, sigma) learning parameters to use for SVM
%with RBF kernel

% You need to return the following variables correctly.
C = 1;
sigma = 0.3;

maxError = Inf;

% It would be nice if this didn't do function optimization by exhaustive search :-/
for currC = [0.01 0.03 0.1 0.3 1 3 10 30]
  for currSigma = [0.01 0.03 0.1 0.3 1 3 10 30]
    model = svmTrain(X, y, currC, @(x1, x2) gaussianKernel(x1, x2, currSigma));

    predictions = svmPredict(model, Xval);
    predictionError = mean(double(predictions ~= yval));

    if predictionError < maxError
      maxError = predictionError;
      C = currC;
      sigma = currSigma;
    end
  end
end

end
```

processEmail.m

```matlab
function word_indices = processEmail(email_contents)
%Preprocesses the body of an email and returns a list of indices of the
%words contained in the email.

% Load Vocabulary
vocabList = getVocabList();

% Init return value
word_indices = [];
```

```

% ========================== Preprocess Email ===========================

% Find the Headers ( \n\n and remove )
% Uncomment the following lines if you are working with raw emails with the
% full headers

% hdrstart = strfind(email_contents, ([char(10) char(10)]));
% email_contents = email_contents(hdrstart(1):end);

% Lower case
email_contents = lower(email_contents);

% Strip all HTML
% Looks for any expression that starts with < and ends with > and replace
% and does not have any < or > in the tag it with a space
email_contents = regexprep(email_contents, '<[^<>]+>', ' ');

% Handle Numbers
% Look for one or more characters between 0-9
email_contents = regexprep(email_contents, '[0-9]+', 'number');

% Handle URLS
% Look for strings starting with http:// or https://
email_contents = regexprep(email_contents, ...
                           '(http|https)://[^\s]*', 'httpaddr');

% Handle Email Addresses
% Look for strings with @ in the middle
email_contents = regexprep(email_contents, '[^\s]+@[^\s]+', 'emailaddr');

% Handle $ sign
email_contents = regexprep(email_contents, '[$]+', 'dollar');


% ========================== Tokenize Email ===========================

% Output the email to screen as well
fprintf('\n==== Processed Email ====\n\n');

% Process file
l = 0;

while ~isempty(email_contents)

    % Tokenize and also get rid of any punctuation
    [str, email_contents] = ...
       strtok(email_contents, ...
              [' @$/#.-:&*+=[]?!(){},''">_<;%' char(10) char(13)]);

    % Remove any non alphanumeric characters
    str = regexprep(str, '[^a-zA-Z0-9]', '');

    % Stem the word
    % (the porterStemmer sometimes has issues, so we use a try catch block)
    try str = porterStemmer(strtrim(str));
    catch str = ''; continue;
    end;

    % Skip the word if it is too short
    if length(str) < 1
       continue;
    end

    % Look up the word in the dictionary and add to word_indices if
    % found
    % =============================================================
    for i = 1:length(vocabList)
      if(strcmp(str, vocabList{i}))
        word_indices = [word_indices ; i];
      end
    end
    % =============================================================

    % Print to screen, ensuring that the output lines are not too long
    if (l + length(str) + 1) > 78
        fprintf('\n');
        l = 0;
    end
    fprintf('%s ', str);
    l = l + length(str) + 1;

end

% Print footer
fprintf('\n\n========================\n');

end
```

emailFeatures.m

```
function x = emailFeatures(word_indices)
%Takes in a word_indices vector and
%produces a feature vector from the word indices.

% Total number of words in the dictionary
n = 1899;

% You need to return the following variables correctly.
x = zeros(n, 1);

for i = 1:length(word_indices)
  x(word_indices(i))=1;
end

end
```